Big Data Mining and Analysis Project 2024

Book Ratings and Reviews Analysis

Objective:

Scrape data from Books to Scrape, a simulated bookstore for web scraping practice, and perform an analysis to compare book ratings, analyse genre trends, and explore pricing strategies.

Prerequisites:

- Basic understanding of Python, HTML, and SQL.
- Python environment with **beautifulsoup4**, **requests**, **sqlite3**, **pandas**, **matplotlib**, and **seaborn** installed.
- Familiarity with Jupyter Notebook.

Task Overview:

1. **Data Collection:**
    - Scrape book data from Books to Scrape.
2. **Data Storage:**
    - Store the scraped data in an SQLite database.
3. **Data Analysis:**
    - Perform analysis using SQL and Python to compare ratings, prices, and explore genre trends.

First Steps:

Step 1: Setting Up Your Environment

- Open Jupyter Notebook and import necessary libraries (**requests**, **BeautifulSoup**, **sqlite3**, **pandas**, **matplotlib**, **seaborn**).
- Install any missing libraries using **pip**.

Step 2: Data Collection

- **Books to Scrape Scraping:**
    - Use **requests** to fetch the book catalogue pages from Books to Scrape.
    - Parse the pages with **BeautifulSoup** to extract book titles, ratings, availability and prices.
    - Store the data in a list or dictionary.

Step 3: Data Storage

- **SQLite Database Setup:**
    - Create an SQLite database **books.db**.
    - Define tables for storing book information, including titles, ratings, genres, and prices.
- **Data Insertion:**
    - Insert the scraped data into the respective tables.

Step 4: Data Analysis

- **SQL Queries for Analysis:**
    - Write SQL queries to analyse the data, such as calculating average prices per genre, distribution of ratings, etc.
- **Python for Advanced Analysis:**
    - Use **pandas** for data manipulation and aggregation.
    - Visualize the results using **matplotlib** and **seaborn** for better understanding.

Second Steps

Task 1: Pagination Handling

Objective: Enhance the scraping script to navigate through multiple pages of the "Books to Scrape" website.

Details: Most websites with a large number of items display them across multiple pages. Modify your script to loop through several pages, scraping data from each page.

Challenge: Identify the pattern in the URL for different pages or locate the 'Next' button link and use it to iterate through pages.

Task 2: Detailed Book Information

Objective: Scrape detailed information for each book.

Details: Instead of just scraping the book list, follow the link to each book's detail page and scrape additional information like the book description, UPC, product type, and availability.

Challenge: Extract and handle the URL for each book's detail page and make additional requests to these pages.

Task 3: Data Cleaning and Preprocessing

Objective: Clean and preprocess the scraped data.

Details: After scraping, the data often contains extra spaces, newline characters, or irrelevant text. Augment a script to clean and format the data appropriately before storing in the database.

Challenge: Develop functions to clean text data, convert prices to a numerical format, and standardize date formats.

Deliverables:

- A Jupyter Notebook containing all code, comments, and analysis.
- An SQLite database with the scraped data.

Task 4 Visualization Task

Objective:

Deepen the data analysis by incorporating detailed Seaborn plots to visualize various aspects of the book data, such as rating distributions, price comparisons, and genre trends.

Details:

- Utilize Seaborn, a Python visualization library, to create insightful plots that uncover relationships and trends within the book data.
- Aim for visualizations that are both informative and visually appealing to convey your findings effectively.

Some starting ideas for Seaborn Plots:

1. **Rating Distribution Plot:**
    - Visualize the distribution of book ratings to identify the most common ratings and overall rating trends.
2. **Price Comparison Plot:**
    - Compare book prices across different rating categories to see if there's a correlation between book ratings and prices.
3. **Genre Popularity Plot:**
    - Assuming genre data is available, show the number of books per genre to highlight the most popular genres.
4. **Price vs. Rating Scatter Plot:**
    - Explore the relationship between book prices and ratings, and include a regression line to identify any trends.

Instructions for Each Plot:

1. **Rating Distribution Plot:**
    - Use **sns.countplot()** with the **rating** column as the x-axis.
    - Customize the plot with appropriate labels and title for clarity.
2. **Price Comparison Plot:**
    - Ensure the price column is in a numeric format for accurate comparison.
    - Use **sns.boxplot()** with **rating** as the x-axis and **price** as the y-axis to visualize price distribution across ratings.
3. **Genre Popularity Plot:**
    - Use **sns.barplot()** with genre data on the x-axis and the count of books on the y-axis.

    - Calculate the count of books per genre using Pandas before plotting.

4. **Price vs. Rating Scatter Plot:**
    - Use **sns.scatterplot()** with **rating** on the x-axis and **price** on the y-axis to visualize the data points.
    - Add a regression line using **sns.regplot()** to highlight any linear trends between price and rating.

Visualization Deliverables:

- **A Jupyter Notebook** containing all code, comments, and analysis. This notebook should include:
    - The web scraping scripts used to collect data from "Books to Scrape."
    - SQL queries performed for data analysis.
    - The Seaborn plots created to visualize the findings from the analysis.
- **Insights and Observations:** Within the notebook, provide detailed observations and insights gained from each plot. Discuss any trends, correlations, or surprising findings that the visualizations reveal about the book data.

Stretch Tasks

Handling Dynamic Content

Objective: Some websites use JavaScript to load content dynamically. Practice scraping a site with dynamic content.

Details: Choose a website that loads its content dynamically (e.g., Goodreads, a news site). Use tools like Selenium to interact with the website and scrape the required data.

Challenge: Identify the elements that are loaded dynamically and write a script using Selenium to scrape data from such elements.

Asynchronous Scraping

Objective: Implement asynchronous requests to speed up the scraping process.

Details: When scraping multiple pages or making numerous requests, use asyncio with aiohttp in Python to send asynchronous HTTP requests.

Challenge: Modify the scraping script to handle asynchronous calls and manage the responses.